# Programming 101

## Learn to Code with the Processing Language Using a Visual Approach

*Second Edition*

Jeanine Meyer

Apress®

# Programming 101

## Learn to Code with the Processing Language Using a Visual Approach

## Second Edition

**Jeanine Meyer**

Apress®

*Programming 101: Learn to Code with the Processing Language Using a Visual Approach*

Jeanine Meyer
Mt Kisco, NY, USA

ISBN-13 (pbk): 978-1-4842-8193-2 ISBN-13 (electronic): 978-1-4842-8194-9
https://doi.org/10.1007/978-1-4842-8194-9



Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: James Robinson-Prior
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 NY Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on the Github repository: https://github.com/Apress/Programming-101. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

*To my family, who inspire and teach me.*

# Table of Contents

# About the Author

**Jeanine Meyer** is Professor Emerita at Purchase College/SUNY. Before Purchase, she taught at Pace University and prior to that was a manager and research staff member at IBM Research in robotics and manufacturing. She also worked as a research consultant at IBM for educational grant programs.

She was moved to create this book because of a general wish to make programming less mysterious and more appealing while featuring the challenges. She enjoys spending time with favorite pictures and video clips as well as producing programs. The chance for a new edition provided a reason to explore p5.js, tools for using JavaScript with features from Processing.

She is the author of five books and coauthor of five more on topics ranging from educational uses of multimedia, programming, databases, number theory, and origami. She earned a PhD in computer science at the Courant Institute at New York University, an MA in mathematics at Columbia, and an SB (the college used the Latin form) in mathematics from the University of Chicago. Recently, she has given lectures, in-person and remotely, connecting origami, mathematics, and computer science as well as the use and misuse of math in the news. She is a member of Phi Beta Kappa, Sigma Xi, the Association for Women in Science, and the Association for Computing Machinery. Jeanine is trying but remains a beginner at Spanish and piano.

# About the Technical Reviewer

**Joseph McKay** is an associate professor of new media. He primarily teaches new directions in virtual space, programming for visual artists, intro to physical computing, hacking the everyday, senior seminar, and web development.

Joe's work is focused on interactive art games. He makes games that have their roots in fine art but are also fun and easy to play. He is currently working on a VR art game with innovative locomotion.

# Acknowledgments

Much appreciation to the subjects of the illustrations in this book, starting with my father (Joseph) and including my mother (Esther), Aviva, Grant, Liam, and especially Annika. Thanks to my children, Aviva and Daniel, for the photography, video, and computer graphics work.

My students, teaching assistants, and colleagues always provide ideas, stimulation, feedback, and advice. Thanks especially to Irina Shablinsky for her efforts in teaching me Processing and how to teach Processing and introducing me to Takashi Mukoda. Thanks to David Jameson, whose comments and concerns made me produce the "Under the Covers" section for each chapter.

Thanks to the crew at Apress/Springer Nature, including for the second edition James Robinson-Prior, Jessica Vakili, Dulcy Nirmala, Krishnan Sathyamurthy, and others I do not know by name. Much appreciation to the past technical reviewers, Massimo Nardone and Takashi Mukoda, and the technical reviewer for this edition, Joe McKay, who brought his considerable talent and experience to the task.

# Introduction

Processing is a programming language built on top of another programming language called Java. To quote from the https://processing.org page, "Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts." The term for a program in Processing is *sketch*. However, Processing can be used to create applications that are far more than static sketches. You can use Processing to create dynamic, interactive programs. It is a great tool for learning programming.

Though Processing was created for visual artists, it serves a broad population of people. In particular, at Purchase College/SUNY, Processing has been an excellent first computer programming language for our computer science/mathematics majors and minors. It also serves students across the college, who take our CS I course to satisfy one of the general education requirements. This experience has been reported in other places. Processing and this text also are appropriate for self-study.

The ten chapters in this book share a common design and structure. My goal is to introduce you to programming, focusing on the Processing language. In each chapter, I explain general programming concepts and specific Processing features through the use of one or more specific examples. The code and files such as image files are combined as zip files and available at https://github.com/Apress/Programming-101. I hope the examples are entertaining; the goal, however, is not for you to learn the specific examples but instead understand the concepts and features. The way to learn programming is to make these examples "your own" and to go on to do a lot of programming.

The introduction to each chapter starts with a brief description of the concepts and programming features used and the examples; then you need to be patient while I provide background. Each chapter includes a discussion of general "Programming Concepts" prior to plunging into the details. These are not limited to the Processing language but are present in most programming languages. Presenting the concepts in a general way might help you if you are coming to this book knowing another language *or* you hope to move on to another language someday.

Next, I describe the "Processing Programming Features" that are used to realize those concepts and produce the examples. This section will have actual code in it and maybe short examples. This is a spiral approach, going from the general to the specific.

A section called “Under the Covers” describes what Processing is doing for us behind the scenes and the relationship between Processing and Java. This section appears in different places in each chapter. It might be of more interest for readers who know something or want to know something about Java, but I urge everyone to give it at least a quick scan.

I then provide an overview of each example, with screenshots showing the operation of the program. Please note that in some cases, I have modified the programs to obtain the screenshots. I then go on to describe the implementation of the example, which contains a **“**Planning” and a **“**Program” section. The “Planning” section is where I describe my thought process. Programs do not spring into existence—at least for me—not like Mozart composing a symphony, which was said to emerge all at once from his mind. It is an iterative process for most of us. This section contains a table indicating the relationship of the functions. The “Program” section includes a table with one column for code and another column with an explanation of that line of code. These tables are long and are not meant to be read as poetry or fine literature. Instead, skip around. Use the function relationship table. If you download the code and try it out, you can use this section to improve your understanding of the program. The most critical step is to make changes, and I provide suggestions in the **“**How to Make This Your Own” section. This set of sections is repeated for each example.

A section titled **“**Things to Look Up” will contain a list of Processing features related to the ones described in the chapter. Processing is a large language, and it is growing. I can show you only a small subset of the features, and each feature is used in one way, perhaps using default values. You can and should consult other references to learn more. You can look things up in multiple ways. For example, you can go to the website at `https://processing.org/reference/` and just keep that open. Alternatively, if you want to look up how to draw a rectangle in Processing, it can be efficient to enter “processing.org rectangle” into Google (or another search engine) or the address field of browsers such as Chrome to retrieve a list of possible sites. It is best to use “processing.org” because “processing” is a common English word. You can try “Processing rectangle,” but you will need to skip over some sites that have nothing to do with the Processing language.

Remember that the goal of this book is not to teach you how to make my examples, from peanut-shaped bald men to my versions of certain games to rotating 3D cubes with photos of my grandchild, but to help you understand how to make your own programs! Make small changes and then large changes. Make your own programs! Chapters will close with two more sections: a brief review, “What You Learned,” and “What’s Next.”

The book also has an Appendix describing what is called p5.js. This is a way to produce programs for the Web by providing a Processing Library to use with JavaScript. The Processing organization also supplies an online editor.

You are welcome to look at the chapters in any order, but later examples do depend on an understanding of concepts introduced earlier. Moreover, because one of the main techniques of programming is to reuse code, there are many instances of later examples copying parts of earlier examples. Do not be concerned: the tables in the "Implementation" section contain complete programs. It is beneficial for your learning process to recognize the repetition.

Please do take a pause in reading to explore, experiment, and, I repeat, make your own programs. Learning how to program is critical for understanding how we function in today's world and the requirements and challenges of devising algorithms using logic and data. Learning to program might help you get a job. However, the main thing that drives me, and I hope will drive you, is that it is fun.

Enjoy,
Jeanine

# CHAPTER 1

# Basics

## Abstract

The goal of this chapter is to get you started. The programming example will be a static drawing of two cartoonish figures, as shown in Figure 1-1. Be aware that the examples in subsequent chapters will increase in complexity, as we will be producing programs that are highly interactive and, possibly, involving random effects, reading files, and exhibiting behavior based on various conditions.




J. Meyer, *Programming 101*, https://doi.org/10.1007/978-1-4842-8194-9_1

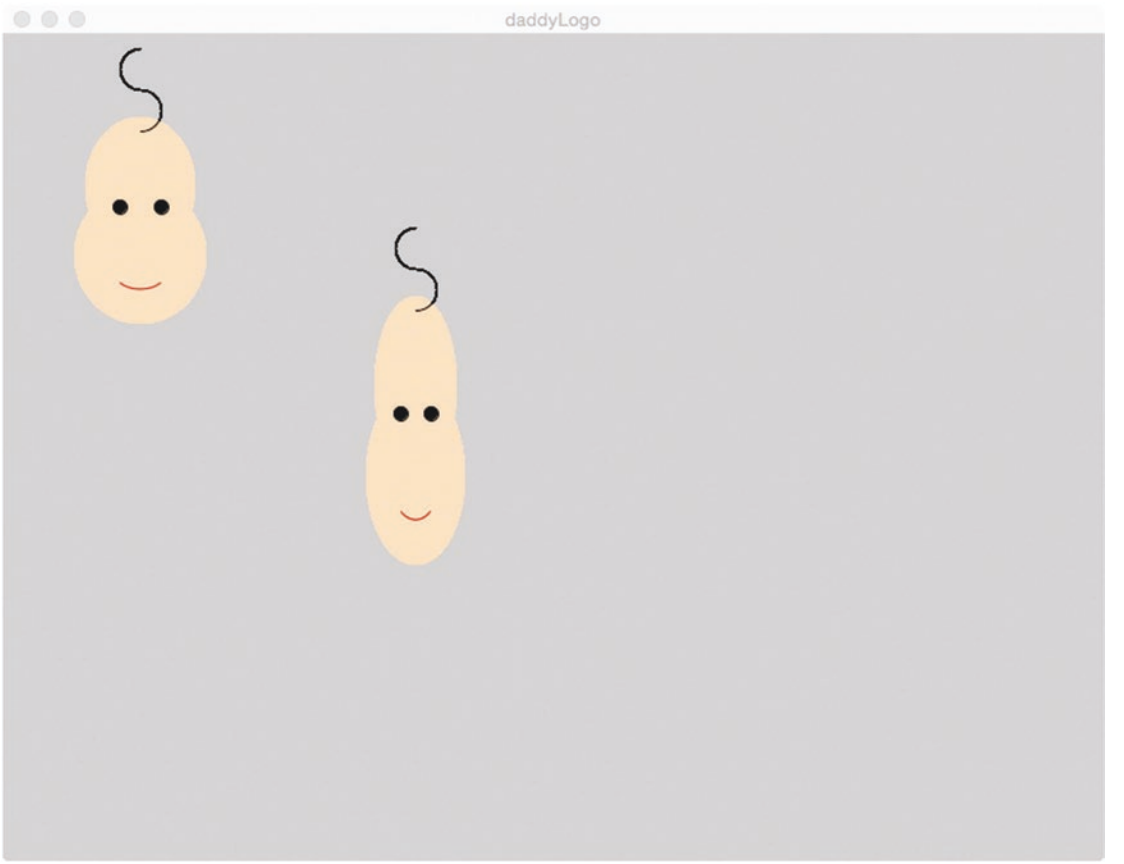


***Figure 1-1.*** *Fat and skinny Daddy logos*

The Daddy logo is a version of a drawing my father would make, often as his signature on a letter or note or artwork. I hope that you will design or recall a drawing or symbol that has meaning to you and makes you happy the same way this cartoonish peanut-shaped, bald guy makes me.

We will need to do some work to start us off and get to the point that the coding is clear, but it is not too difficult. The traditional first task in using any programming language is to get the program to display the phrase "Hello, world." This works well in demonstrating several important concepts, including what happens if the programmer makes certain types of errors. Because of the features built into Processing, you can produce a pretty fancy version of "Hello, world."

Be patient with me and with yourself. At the end of the chapter, you will be able to implement your own Daddy logo.

# Programming Concepts

This section, included in each chapter, is to provide a general introduction to concepts. I begin with comparing and contrasting programming languages with natural languages.

## Programming Languages and Natural Languages

Programming languages have some similarities with natural languages, but they also have significant differences. Programming languages are defined by rules just as a natural language's grammar defines what is proper English, Spanish, or other language. A program contains statements of different types just as we find in English (or Spanish, etc.), and there also are ways to construct compound statements. Statements in programming languages contain terms and expressions involving terms. In programming languages, programmers often come up with our own names for things. The names must follow certain rules, but these are not unduly restrictive. This is a difference from natural languages, in which we mainly use the official words of the language, whereas in programming, we are extending the language all the time.

A more significant difference between programming languages and natural languages is that the rules must be obeyed at all times when using programming languages! Consider that we all frequently utter grammatically incorrect statements when we speak and yet generally are understood. This is not the situation in programming. The good news in the case of Processing, and certain other languages, is that the Processing system generally indicates where an error occurs. The development environments for Processing and other computer languages are themselves computer programs, and they do not exhibit any impatience while we fix errors and try the program again. I will give some examples of statements, right after I introduce the concept of values and variables.

## Values and Variables

Programming involves containers or buckets where we can store specific types of things (values). These kinds (types) of things are called *data types*. The following are some examples of data:

```
Int          integer (e.g., 10)
float        decimal value (e.g., 5.3)
```

```
Boolean     logical values (e.g., true/false)
Char        single character (e.g., 'a')
String      a string of characters (e.g., "hello world")
```

String should start with a capitalized "S". The B in Boolean can be upper or lowercase. The data type is named for George Boole, an English mathematician credited with originating symbolic algebra.

Our programs can include literal values such as 5, 100.345, and "Hello" in the code. In addition, a feature in all programming languages is what is termed *variables*. A variable is a construct for associating a name of our choosing with a value. We can initialize the variable, change it, and use it in an expression; that is, the value associated, often termed *in* the variable, can vary, that is, change. Using variables makes our programs less mysterious. Moreover, we can define one variable in terms of another, making relationships explicit and preventing certain errors. In Processing, Java, and some, but not all, programming languages, variables need to be declared, or set up before use. One characteristic of variables is termed *scope,* which indicates what code has access (e.g., global variables vs. local variables), but that is best explained later.

The following are examples of Processing statements. Explanation is given in comments and later.

```
int classSize;  // this declares, that is, sets up classSize to
                // be a variable.
classSize = 21; //assigns the value 21 to the variable classSize.
classSize = classSize + 5; //takes whatever is the current value held in
               // the variable classSize
               // and adds 5 to it and resets classSize to the new value
float score = 0; //declares the variable score AND
                 // assigns it a value. This is called initialization.
if (score == 0) {
      text("You did not score anything.", 100,100);
      text("Try again.", 100,300);
}
```

The // indicates that the rest of the line is a comment, meaning that Processing ignores it. It is intended for readers of the code, including you, to make things clear. You also can use the delimiters /* and */ for long comments.

## Note

My examples, because they are surrounded by explanations, tend not to have as many comments as I would use outside of teaching and writing books.

There are rules for variable and function names in all programming languages. Generally, they must start with a letter, uppercase or lowercase, and cannot contain spaces. The most important guidance for naming is that the names should have meaning for you. The programming language will accept single character names or names with no apparent meaning, but these will not be helpful when you are trying to recall what you were trying to do. So-called camel casing, as in `classSize`, can be helpful.

A single equal sign (=) means assignment and is used in what are called, naturally enough, *assignment* statements and *initialization* statements. The statement

```
classSize = classSize + 5;
```

will seem less illogical if you read it as

*classSize is assigned or gets the total of the current value of classSize plus 5.*

A double equal sign (==) is a comparison operator and often appears in an `if` statement. Think of it as like < or <=.

The `if` statement is an example of a compound statement. The expression `score == 0` is interpreted as a comparison. If the value of the variable `score` is equal to zero, then the statement within the brackets is executed. If the value of `score` is greater than zero or less than zero, nothing happens. Again, you will see many more statements in the context of examples.

# Functions

Programming work in any language is structured into units. One important way of structuring code comes with different names: *function, procedure, subroutine, method.* These are ways of packaging one or more statements into one unit. You will read about functions in the "Processing Programming Features" section and methods in the "Under the Covers" section. Briefly, functions are defined, and functions are invoked. I can give you directions , perhaps orally, perhaps by text, to my house, which is analogous to defining a function. At that point, I am not directing you to come to my house. At some later time, I can direct you to go to my house, and this is analogous to invoking the function.

Programs can be considerably shorter as well as easier to modify through the use of functions and variables, so understanding both of these concepts is important. You do not need to accept this or understand this right now. It will be demonstrated later by my sketch for displaying two Daddy logos that takes just one statement more than displaying the Daddy logo just once.

## Specifying Positions and Angles

Displaying drawings and images and text on the screen requires a coordinate system. The coordinate system used by most computer languages and many graphical tools is similar to what we learned (but might or might not remember) from high school geometry, with one big difference. Horizontal positions, sometimes called *x* positions, are specified starting from the left. Vertical positions, sometimes called *y*, are specified starting from the top of the screen. Figure 1-2 shows the coordinate system with a small circle at the 100, 200 location.

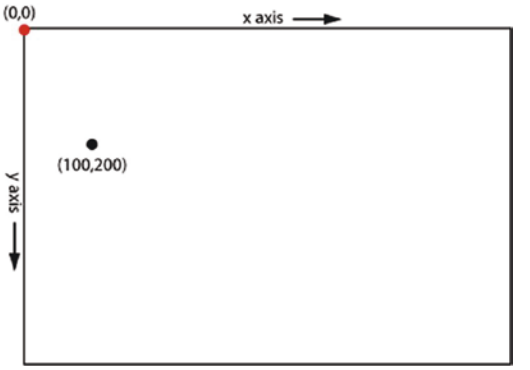


***Figure 1-2.*** *Coordinate system*

If you say to yourself "This is upside down," then I know you understood. Another important point is that the unit is very small, so if your code positions something at 100, 200 and later at 101, 201, you probably will not detect the difference. Your intuition regarding this will improve with experience.

## Note

As a teaser, Processing has facilities for 3D as well as 2D. We get to 3D in later chapters.

In this chapter, my Daddy logo has a smile made by specifying an arc of an ellipse. To produce the arc, I need to write code to indicate a starting angle and an ending angle of the arc. The system used in most computer languages is not the standard one in which a right angle is 90 degrees, a U-turn is a 180, and snowboarders do 1800s. (I am writing this during the Olympics, and yes, snowboarders did tricks measuring 1800 and bigger.) It might be upsetting to realize this, but the notion of degrees with a circle consisting of 360 degrees was invented by people. I typically offer my students extra credit to identify where and when this happened. Instead, in most programming languages, we use a measure called *radians*. Think of wrapping a circle with lengths equal to one radius. How many lengths will this take? You know the answer: It is not a whole number, it is 2 times π, where π is an irrational number often approximated by 3.14159. In our programming, we will use the built-in values TWO_PI, PI, HALF_PI, and QUARTER_PI. You will see radians in use, so be patient.

# Colors

There are different ways to specify colors in computer languages and computer applications, and Processing supports more than one. In this text, we stick with grayscale and RGB (red/green/blue). Because of how these values are stored, the range of grayscale is from 0 (black) to 255 (white), and the values for redness, greenness, and blueness are specified by a number from 0 to 255. This approach is used in many applications. If you want to use a certain color that you see in a photo, you can open the image file in Adobe Photoshop or the online Pixlr or some other graphics tool and use the eye drop on the pixel (picture element) you want, and an information window will tell you the RGB value. See also the mention of the Color Selector in the "Things to Look Up" section.

# Development Environment

Programmers need to prepare programs and test programs. We also need to save our work to come back to it another time. We might need to send the program to someone else. Processing has what is termed an *integrated development environment*, the Processing Development Environment (PDE), which provides a way to prepare and

make changes to a program as well as test it and save it. To give you a different example, Hypertext Markup Language (HTML) documents containing JavaScript are prepared and saved using a text editor, such as Sublime. The resulting files are opened (and run) using a browser, such as Chrome. In the Appendix, I will show you how to use an editor for p5.js, which is a version of JavaScript incorporating Processing features.

## Role of Planning

I close this first "Programming Concepts" section by noting that preparing programs such as a Processing sketch generally involves planning and design. It might be best to step away from the keyboard. Some of the plans might need to be modified when you get to writing the code, but it is best to have plans!

# Under the Covers

As I indicated earlier, Processing is a language built on Java. This means that the Processing code you write is Java code that the development environment puts into a larger Java program prepared for handling Processing sketches. In Java, there are no functions, but, instead, what are termed *methods*. I will introduce methods for our use in Processing in Chapter 4.

The PDE (Processing Development Environment) makes use of *libraries*, collections of methods holding the built-in functions of Processing, such as functions to draw a rectangle.

In the big Java program, there are calls to functions that we write, or, to put it more accurately, we code the body of the function. For example, all Processing sketches contain a function called `setup`, the purpose of which is to do what the name implies. It nearly always includes a statement that defines the width and height of the window, for example. The big Java program invokes the `setup` program once at the start of the sketch. Similarly, we can write the body of a function named `draw`. The Java program invokes this function over and over, the frequency defined by the *frame rate*, which can be reset by assigning a value to the built-in variable `frameRate`. This enables us to build applications producing animations and responding to events such as a user clicking the mouse button. There are many other functions for which we, the programmers, specify the response to an event, for example, `keyPressed` or `mouseClick`.

The Java program also defines *default* settings. Processing and other computer languages and many computer applications provide powerful features. If we needed to specify each aspect of each feature before anything happens, it would be tremendously burdensome. It is important to be aware that certain things can be adjusted, though, as you will see in our very first example later, with the discussion on default values for font, text size, fill color, and stroke color.

The design and capabilities of Processing provide us a way to get started creating and implementing our ideas quickly.

# Processing Programming Features

In this section, I explain the concepts focusing on Processing features. There will be small coding examples to prepare for the larger (although not too large) examples covered later in the chapter.

To use Processing, you need to go to the `processing.org` website and follow the directions to download and install Processing on your computer.

## Processing Development Environment

To describe the PDE in abstract terms is too difficult, so let's get started. Once you have downloaded and installed Processing, open it. At the top of the PDE window, you will see the Processing File toolbar.

Click File, which will open a drop-down menu. Select New. The toolbar will change to hold more options. A window that looks like Figure 1-3 will appear on your screen. The number after sketch_ will be different than what you see here. I believe in saving early, and often so, at this point, you can think about where you want to save your Processing work in terms of the file system on your computer. I leave that to you. You also should give some thought to what you will name each sketch. I suggest the name `first0` for this one. Click File, then select Save As…, and proceed with a file name and a location in the usual way for your operating system.

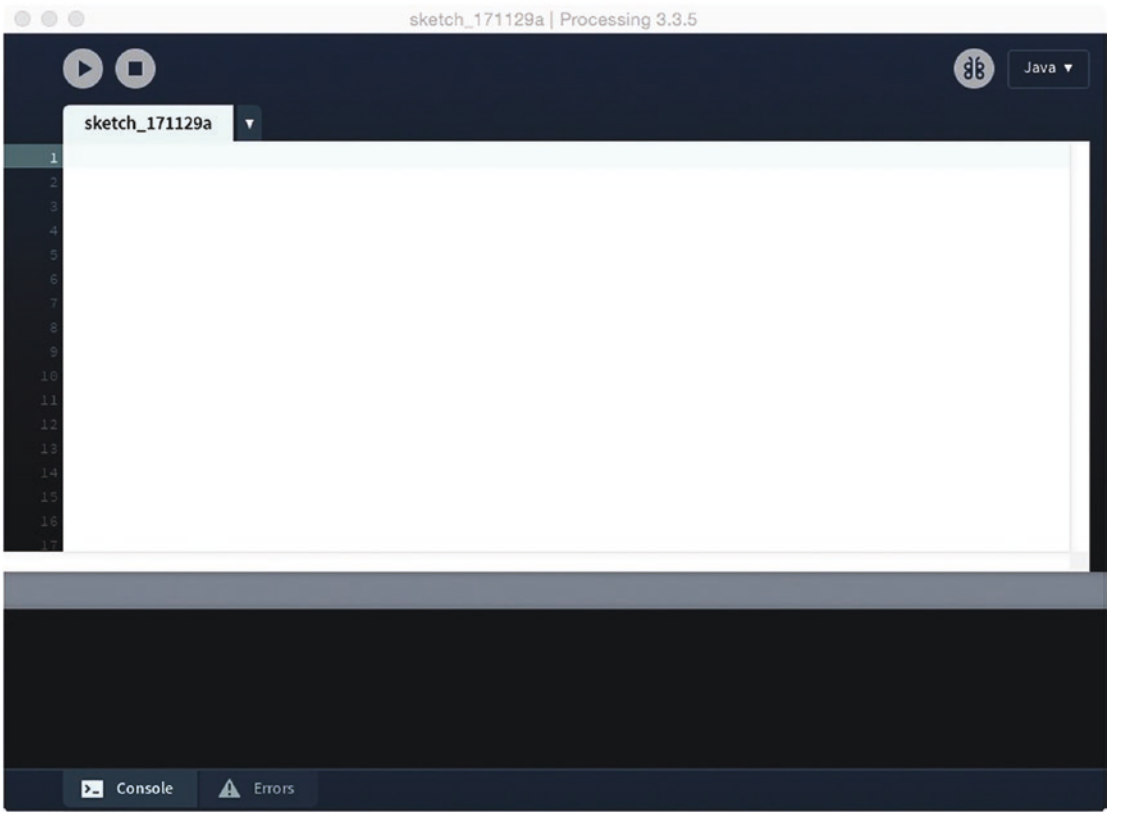


***Figure 1-3.*** *Window for new sketch*

Using Save As… in the PDE produces a folder, in this case named `first0`, which contains a file named `first0.pde`. The examples explored in future chapters will consist of folders containing additional items. For example, a Processing sketch named `myFamily` that makes use of an image file `aviva.jpg` and an image file `daniel.jpg` will be a folder named `myFamily` containing a file named `myFamily.pde` and a folder named `data` that contains the two files `aviva.jpg` and `daniel.jpg`. The relationship of these files is shown in Figure 1-4.

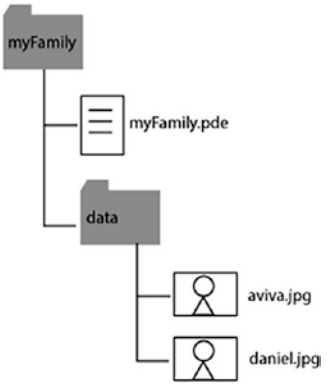


***Figure 1-4.*** *Typical file structure for a sketch*

## Functions

Processing uses the term *function* for grouping together one or more statements into something that can be invoked (called). Functions are defined with *header* statements and then the *body*, a sequence of statements, contained within brackets. You will see in this chapter and every chapter definitions for the `setup` function, a function that Processing expects the programmer to supply. The header is

```
void setup()
```

The term *`void`* indicates that this function does not produce or return a value. The opening and closing parentheses with nothing between them indicate that this function does not expect any parameters.

The Daddy logo example includes a function called `daddy` that does the work of drawing the cartoon. Its header is

```
void daddy(int x, int y, int w, int h)
```

The parameters are the things between the parentheses. The parameter list is the place for the programmer to give names and specify the data type. This means that when I wrote the code to invoke daddy, which is necessary because daddy was something I made up, not anything Processing expects, Processing will check that the values cited in the call are the correct type.

I feel obliged to show you an example of a function that does produce a value, a standard one supplied in many textbooks.